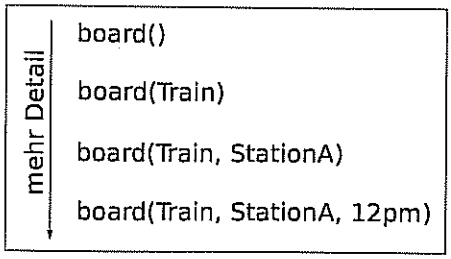

Abb. 3: Aufruf der Funktion board: Mehr Parameter = Mehr Detail

ihre Informationen als auch ihre Topologie erhalten.

Die Interaktion zwischen Objekten und Prozessen muss berücksichtigt werden. Zum Einen können Prozesse den Zustand von Objekten verändern. Zum Anderen sind Objekte aber auch oft essentieller Bestandteil von Prozessen (Prozessträger).

3 Beispielprozess mit schematischer Formalisierung

3.1 Routenbeschreibungen in unterschiedlichen Detailstufen

Folgende unterschiedlich detaillierte Beschreibungen könnten die Fahrt vom Hauptbahnhof Wien zum Flughafen Wien beschreiben (Routenbeschreibung als Prozess). Solche Beschreibungen können z. B. verbal erfolgen (wie in unserem Beispiel) oder mit Hilfe eines Informationssystems (Fahrplanauskunft) erstellt werden. Die dargelegten Detailstufen sind willkürlich gewählt und dienen lediglich der Anschauung.

Level 0 – Dieses Level gibt keinerlei Auskunft über die Abfahrtszeit, Verkehrsmittel oder etwaige Umsteigehalte: *„Um schnell zum Flughafen zu kommen, fährst du am Besten über den Hauptbahnhof"*. Eine solche Beschreibung hat natürlich nur eine Berechtigung, wenn der Angesprochene zu Beginn nicht mehr Informationen benötigt, da er sich z. B. die Informationen während der Reise sukzessive aneignet. Norman (1988) unterscheidet z. B. zwischen „information in the world" und „information in the head". Das erste Konzept bezieht sich auf Informationen, die nicht explizit im Kopf gespeichert sind, sondern die sich der Nutzer aus der Umwelt holen kann (z. B. mit Hilfe von Wegweisern). Das zweite Konzept bezieht sich auf die Informationen, die explizit vom Nutzer gespeichert sind, d. h. auf Wissen über die geographische Umgebung.

```
data Mode = Train | Subway | Bus

board :: Mode -> station -> time -> stateChange World

boardTrain :: place -> time -> stateChange World
boardTrain = board Train

boardTrain_StationA :: time -> stateChange World
boardTrain_StationA = boardTrain stationA

boardTrain_StationA_12pm :: stateChangeWorld
boardTrain_StationA_12pm = boardTrain_StationA "12pm"
```

Abb. 4: Ein Codierungsbeispiel in der funktionalen Programmiersprache Haskell

Level 1 – Dieses Level gibt zusätzliche Auskunft über etwaige Umsteigehalte und Verkehrsmittel, lässt aber Abfahrtszeiten außer Acht. *„Steig in den Zug S1 am Hauptbahnhof, fahre bis zur Station X und wechsle dort in die U-Bahn U2, steige dann am Flughafen aus"*. Diese Beschreibung setzt voraus, dass der Reisende entweder zeitlich flexibel ist, oder einen genügend großen Puffer hat und ihn damit die genauen Abfahrtszeiten nicht interessieren.

Level 2 – Dieses Level gibt zusätzlich Abfahrtszeiten an. *„Nimm den Zug S1 am Hauptbahnhof um 12 Uhr, [...], nimm die U-Bahn U2 um 12.15, [...], steige am Flughafen aus"*. Dieses Level impliziert eine zeitliche Beschränkung, z. B. muss der Reisende in den Zug um 12 Uhr einsteigen, sonst wird das Flugzeug verpasst.

Level 3 – Dieses Level führt zusätzliche Transportmodi ein (z. B. Fußgänger), die unter Umständen anders modelliert werden müssen. *„Um den Zug um 12 Uhr am Hauptbahnhof zu erreichen, musst du um 11.45 Uhr an deiner Wohnung losgehen, [...], sei spätestens um 13.00 Uhr am Flughafen, damit du noch Zeit hast, um zum Abfluggate zu gehen"*. Ein Detailgrad der folgenden Form ist in der allerseltensten Fällen notwendig: *„[...] Betrete die Station, kaufe dir eine Fahrkarte, gehe zum Gleis 1, warte, bis der Zug S1 eingefahren ist, öffne die Tür, steig ein, warte bis zu nächsten Station, entscheide ob du aussteigen musst, [...]"*.

Die Information, dass man ein Ticket kaufen muss, um in den Zug einsteigen zu können (Dies ist allerdings nur eine soziale, keine physische Einschränkung), ist implizit in der Form von sogenannten „scripts" gespeichert. Scripts sind kognitive Wissensstrukturen, die wir durch Erfahrung mit unserer Umwelt generieren und die uns helfen, kulturell akzeptierte Handlungsweisen auszuführen (Schank und Abelson, 1977). Das „restaurant script" sagt uns z. B., welche Aktionen wir in einem Restaurant durchführen können (z. B. Essen bestellen, aber nicht Fußball spielen) und welche Rollen von welchen Personen eingenommen werden können (z. B. Ich als Kunde bestelle bei einer Person, genannt Kellner). In Fällen, bei denen Routenbeschreibungen für Nutzer mit besonderen Bedürfnissen erstellt werden sollen, z. B. Kinder (weniger Erfahrung) oder Ausländer (kulturelle Unterschiede), kann ein solcher Detailgrad allerdings nützlich sein.

3.2 Schematische Formalisierung

Betrachten wir die Beispiele aus 3.1, stellen wir Folgendes fest: Der „gleiche" Prozess existiert über mehrere Detailstufen hinweg. In der Beschreibung treten sowohl Prozesse (z.B. Einsteigen) als auch Objekte (z. B. Zug, Station, Uhrzeit) auf, die miteinander interagieren.

Prozesse werden detaillierter, umso mehr Objekte zur Verfügung stehen (siehe Abb. 3). Beispielsweise ist „Einsteigen in die U-Bahn", da überall möglich, genereller als „Einsteigen in die U-Bahn am Hauptbahnhof".

Die grundlegende Idee ist nun, die Prozesse und die beteiligten Objekte als Funktionen bzw. als Argumente der Funktionen vollständig zu modellieren (in der höchsten Detailstufe) und je nach gewünschtem Detailgrad nur partiell auszuwerten. Um das Konzept der partiellen Funktionsauswertung zu veranschaulichen, sei das folgende Beispiel genannt. Die Funktion add mit der Funktionssignatur add :: Int → Int → Int und der Funktionsdefinition add x y = x + y gibt das Resultat (eine Zahl) der Addition von zwei ganzen Zahlen zurück. Wird der Funktion allerdings nur ein Argument übergeben, z. B. die ganze Zahl 1, erhalten wir als Resultat eine neue Funktion, die eine ganze Zahl als Argument entgegen nimmt und diese um 1 erhöht. Das Resultat ist eine Funktionsdefinition add1 = add 1 und eine Funktionssignatur add1 :: Int → Int.

Im nachfolgenden Beispiel sei der Prozess des Einsteigens als Funktion board() beschrieben. Beim Prozess des Einsteigens sind verschiedene Objekte involviert, z. B. das Verkehrsmittel in das eingestiegen